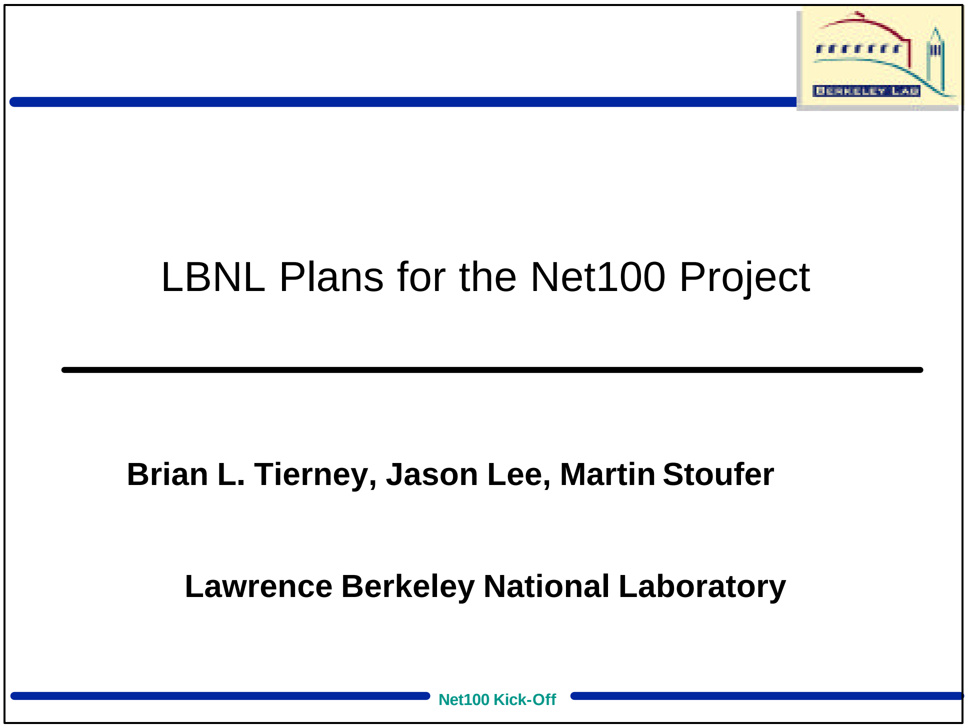
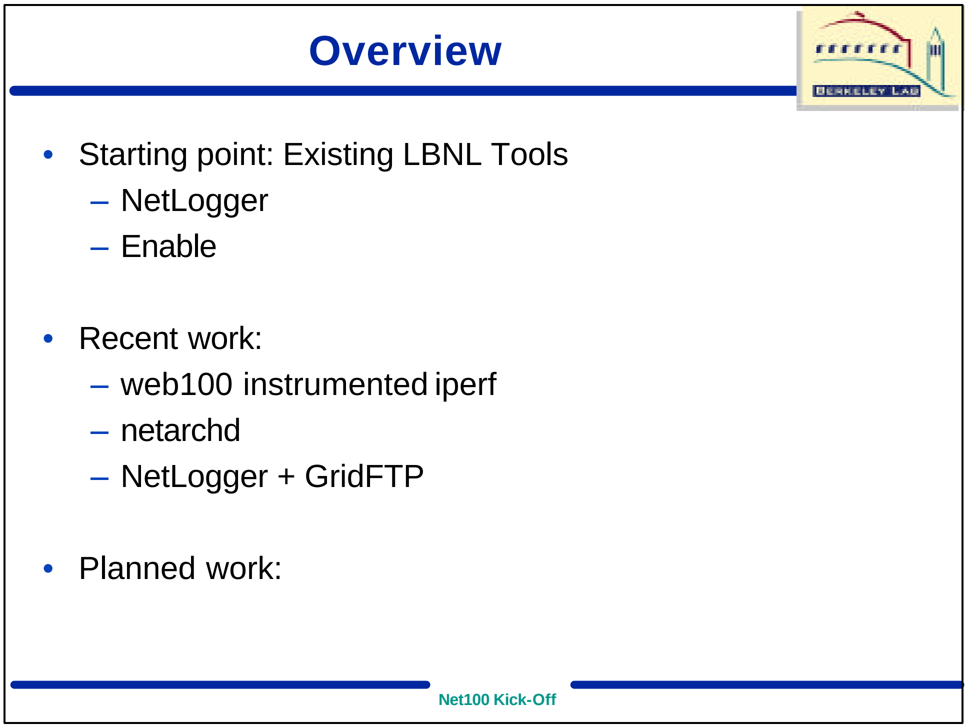


LBNL Plans for the Net100 Project

Brian L. Tierney, Jason Lee, Martin Stoufer

Lawrence Berkeley National Laboratory



Overview

- Starting point: Existing LBNL Tools
  - NetLogger
  - Enable
- Recent work:
  - web100 instrumented iperf
  - netarchd
  - NetLogger + GridFTP
- Planned work:

Network Tool Analysis Framework (NTAF)

- Configure and launch network tools
  - measure bandwidth/latency (*iperf, pchar, pipechar*)
  - collect passive data (SNMP from routers, OS counters)
  - augment tools to report Web100 data
- Collect and transform tool results into a common format
- Save results for short-term auto-tuning and archive for later analysis
  - compare predicted to actual performance
  - measure effectiveness of tools and auto-tuning
- Use NetLogger to format and send data to archive
- Use Enable as starting point for NTAF

NetLogger Overview

NetLogger Toolkit

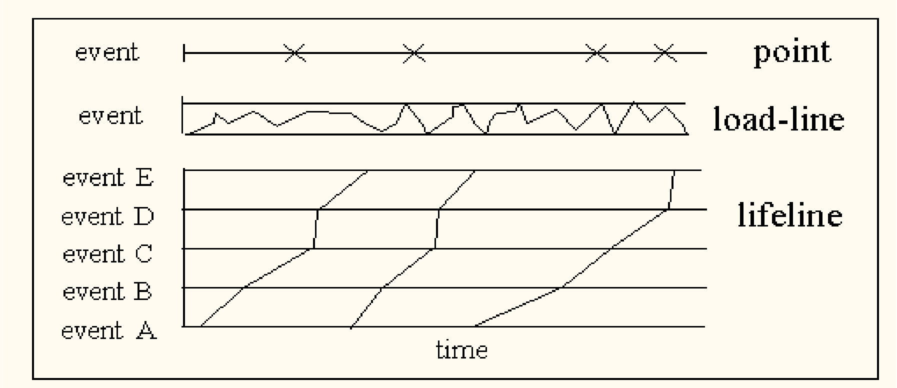
- We have developed the NetLogger Toolkit (short for Networked Application Logger), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
  - tools for host and network monitoring
- The approach is novel in that it combines network, host, and application-level monitoring to provide a complete view of the entire system.
- This has proven invaluable for:
  - isolating and correcting performance bottlenecks
  - debugging distributed applications

NetLogger Components

- NetLogger Toolkit contains the following components:
  - NetLogger message format
  - NetLogger client library (C, C++, Java, Perl, Python)
  - NetLogger visualization tools
  - NetLogger host/network monitoring tools
  - NetLogger storage and retrieval tools (new)
- Source code and binaries are available at:
  - http://www-didc.lbl.gov/NetLogger/
- Additional critical component for distributed application analysis:
  - NTP (Network Time Protocol) is required to synchronize the clocks of all systems

NLV Graph Primitives

- NetLogger visualization tool (nlv) supports graphing of “points”, load-lines, and lifelines



Sample NetLogger Use

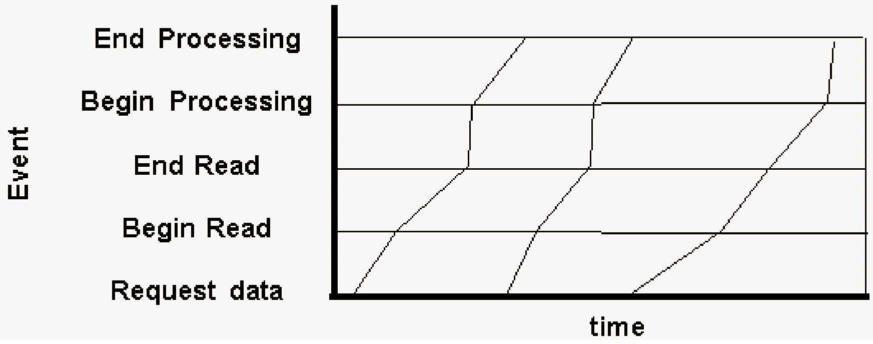
```
logger = NetLogger.NetLogger(progname)
err =logger.nlOpen (x-netlog://loghost.lbl.gov,
                         NetLogger.NL_ENV)

while not done :
   logger.nlWrite ("EVENT_BEGIN", "SIZE=%d" %
     size);
   done = do_something(data, size)
   logger.nlWrite ("EVENT_END", "SIZE=%d" %
     size);
   loop_cnt++

logger.nlClose()
```

NetLogger Open Call

```
NLhandle   *lp = NULL;
lp = NetLoggerOpen(char *program_name,
  char *dest_url, int flags);
```

- Based on *dest_url* and *flags,* can send log messages to:
  - disk
  - memory
  - syslogd
  - netlogd
- can specify these values via environment variables

Sample NetLogger “Life Lines”



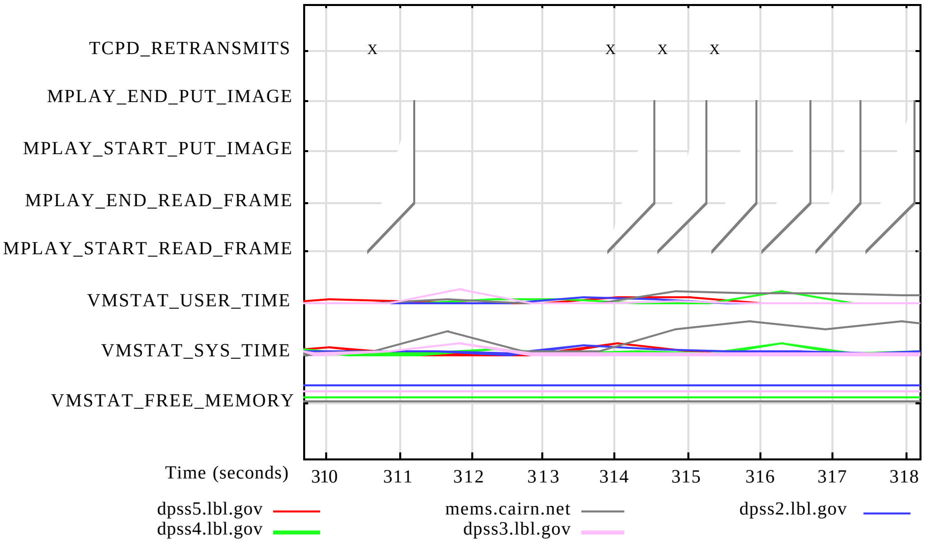
generating Lifelines require all related events to have the same “event ID” to tie events together

example event IDs: Block number, loop counter, etc.

Sample NetLogger Use with Event IDs

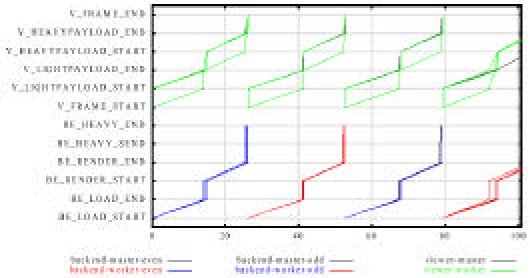
```
lp = NetLoggerOpen(progname, NULL, NL_ENV);
for (id=0; id< num_blocks; id++)  {

   NetLoggerWrite(lp, “START_READ”, “ID=%d SIZE=%d”, id, size);
   read_block(i);
   NetLoggerWrite(lp, “END_READ”, “ID=%d SIZE=%d”, id, size);

   NetLoggerWrite(lp, “START_PROCESS”,“ID=%d SIZE=%d”, id,size);
   process_block(i);
   NetLoggerWrite(lp,“END_PROCESS”,“ID=%d SIZE=%d”, id, size);

   NetLoggerWrite(lp,“START_SEND”,“ID=%d SIZE=%d”, id, size);
   send_block(i);
   NetLoggerWrite(lp, “END_SEND”, “ID=%d SIZE=%d”, id, size);
}

NetLoggerClose(lp);
```

Example: Combined Host and Application Monitoring

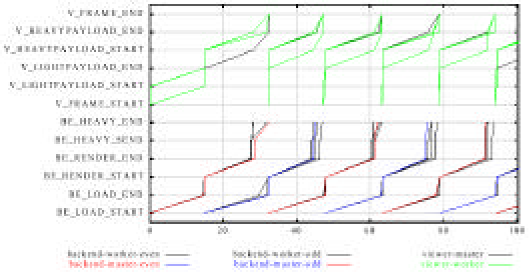


NetLogger Tuning Results

- I/O followed by processing



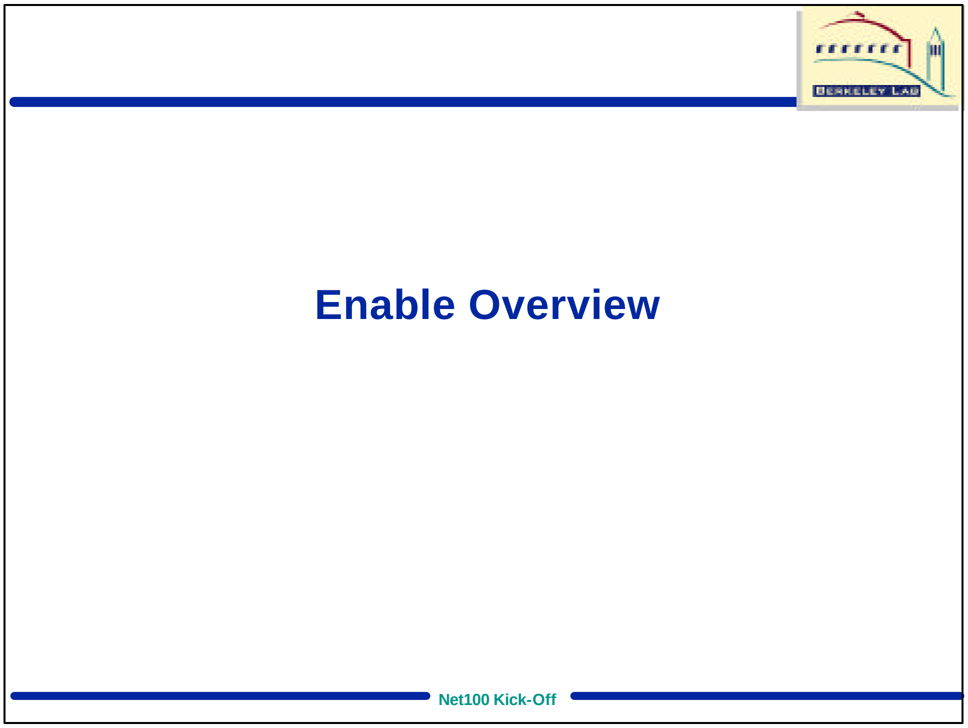
- overlapped I/O and processing



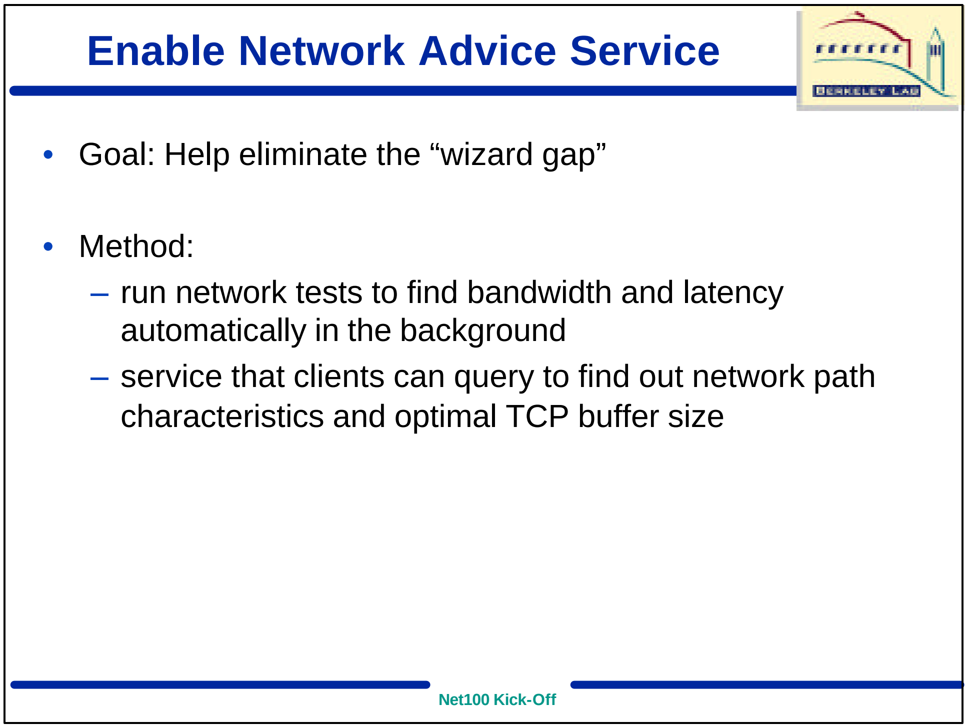
almost a 2:1 speedup

NetLogger Future Work

- support for Binary and XML formats
- Producer / consumer interfaces
  - part of “Grid Monitoring Architecture” work



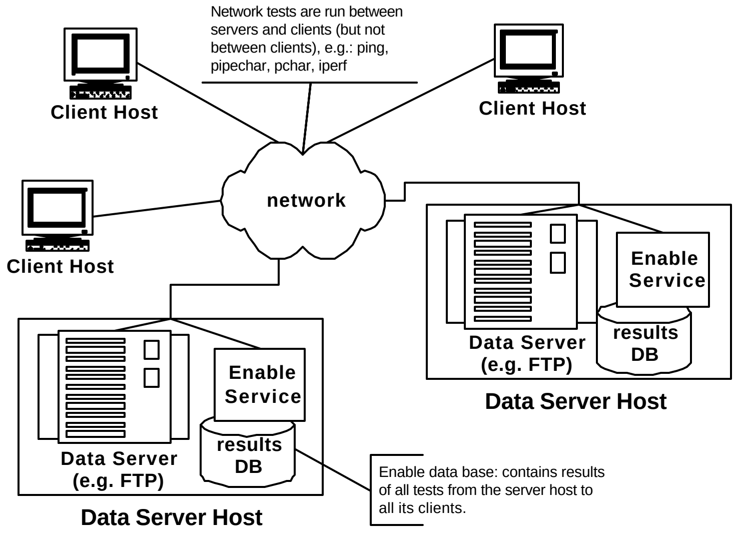
Enable Overview



Enable Network Advice Service

- Goal: Help eliminate the “wizard gap”
- Method:
  - run network tests to find bandwidth and latency automatically in the background
  - service that clients can query to find out network path characteristics and optimal TCP buffer size

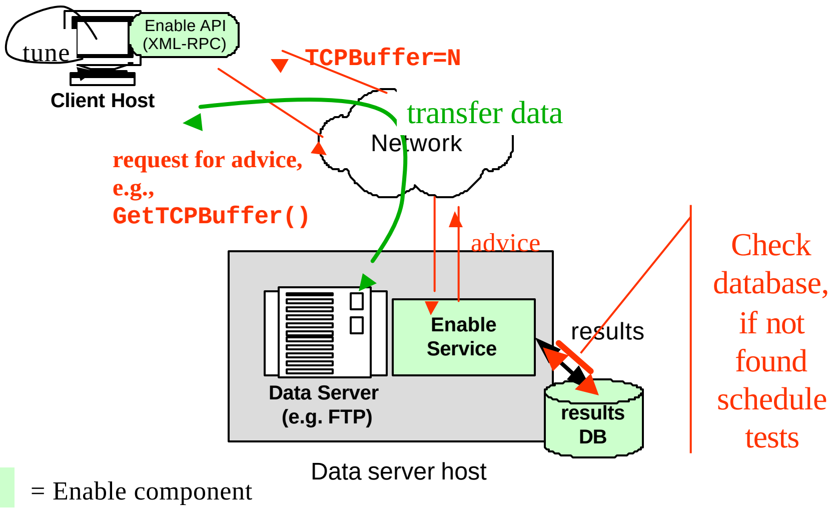
Enable Architecture



Network tests are run between servers and clients (but not between clients), e.g.: ping, pipechar, pchar, iperf

Enable data base: contains results of all tests from the server host to all its clients.

Use-Case (1): Single client

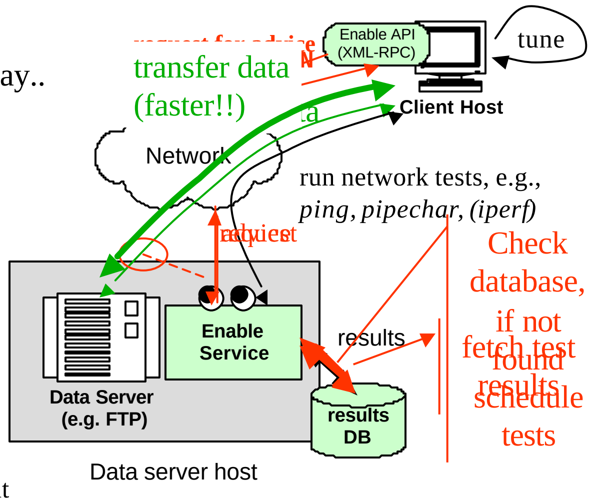


Check database, if not found schedule tests

🟩 = Enable component

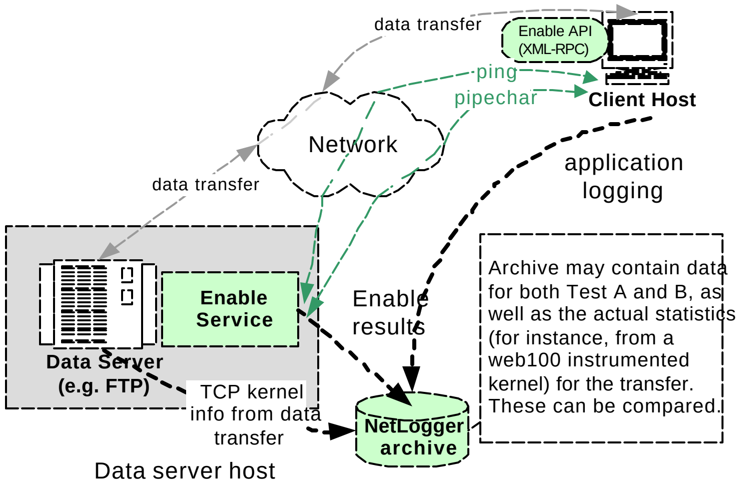
Use-Case (2): Automatic testing

Later that day..



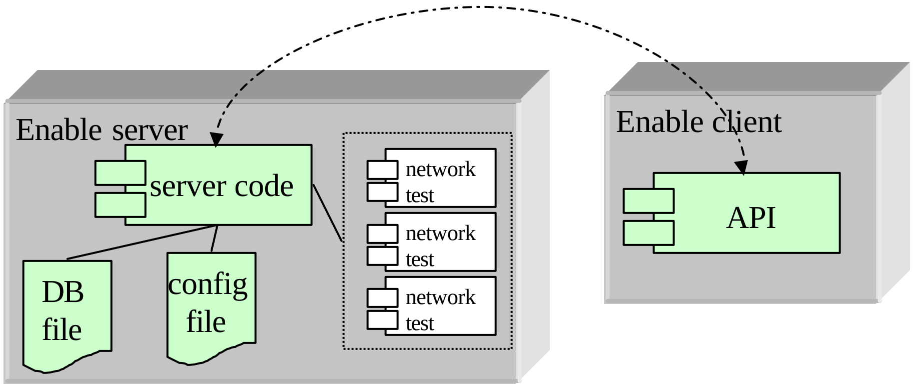
🟩 = Enable component

Use-Case(4): Tool Comparison



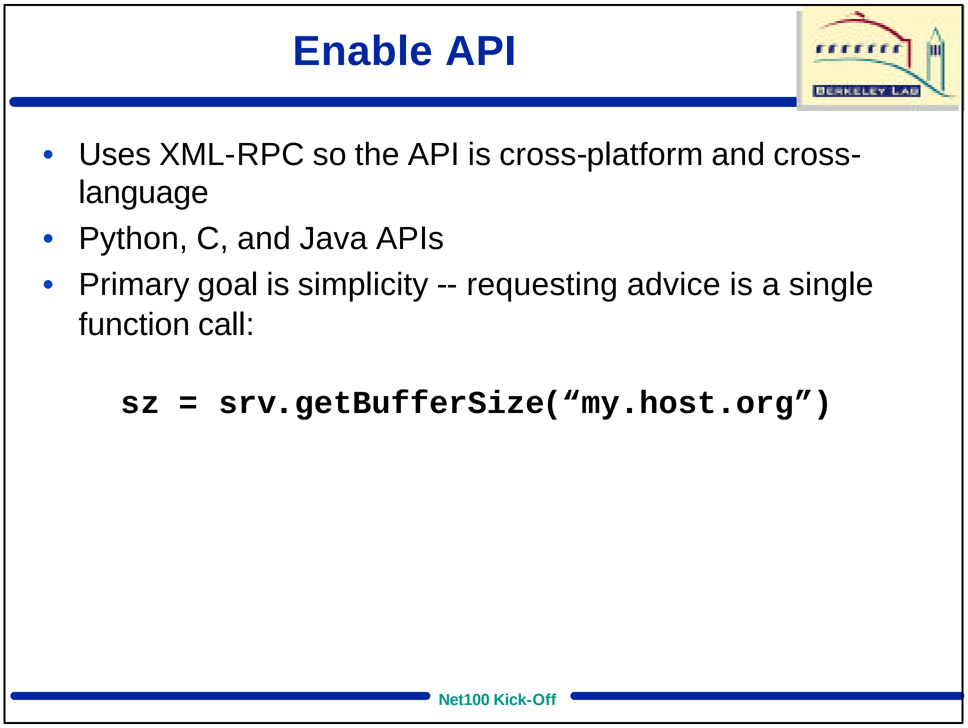
Archive may contain data for both Test A and B, as well as the actual statistics (for instance, from a web100 instrumented kernel) for the transfer. These can be compared.

Enable Components



Enable server functionality

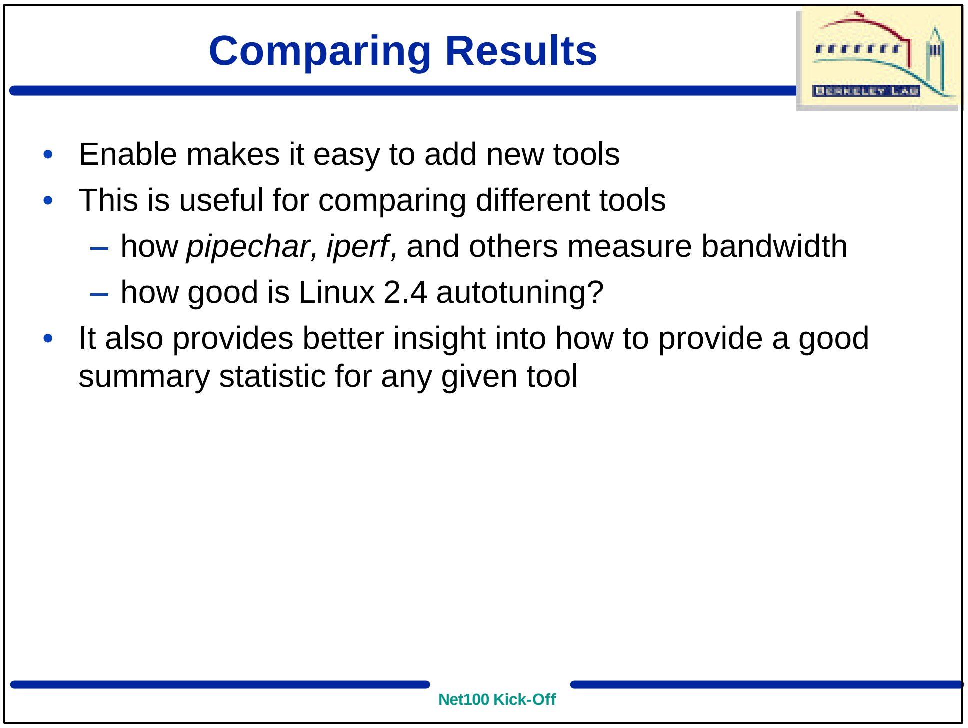
- Schedule tests
  - Current network tests: *ping*, *pipechar*, *iperf*
  - consider conflicts: *iperf*, *pipechar* need to run serially; multiple *pings* can run in parallel.
- Record last *N* results in local database
  - all test results can be sent to *netarchd*
- Perform analysis needed for advice
  - keep running average of all tests to keep track of long-term changes in characteristics
  - do trimmed-mean and stddev of last *N* to calculate TCP buffers, etc.



Enable API

- Uses XML-RPC so the API is cross-platform and cross-language
- Python, C, and Java APIs
- Primary goal is simplicity -- requesting advice is a single function call:

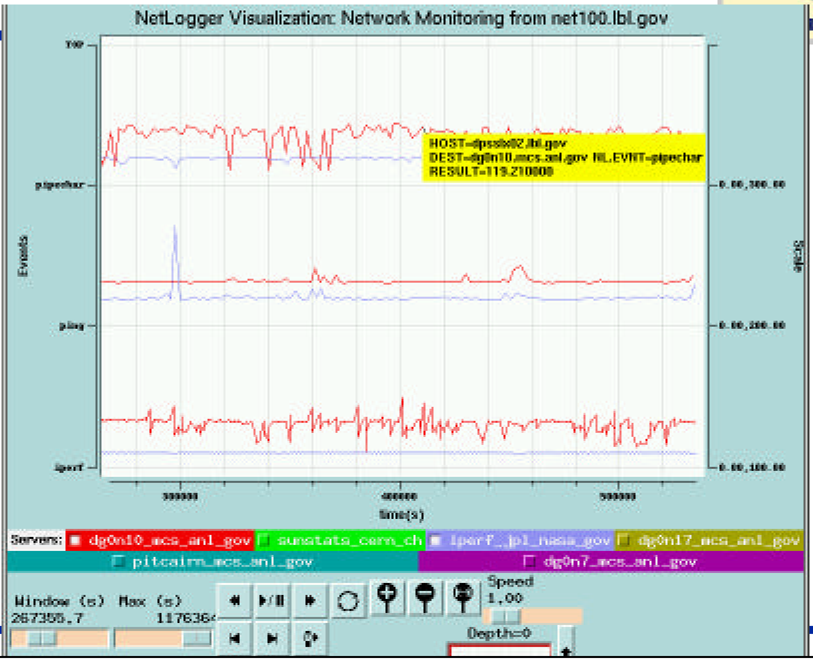
```
sz = srv.getBufferSize(“my.host.org”)
```



Comparing Results

- Enable makes it easy to add new tools
- This is useful for comparing different tools
  - how *pipechar, iperf,* and others measure bandwidth
  - how good is Linux 2.4 autotuning?
- It also provides better insight into how to provide a good summary statistic for any given tool

nlv of Enable Data



HOST=dpsslx02.lbl.gov
DEST=dg0n10.mcs.anl.gov NL.EVNT=pipechar
RESULT=119.210000

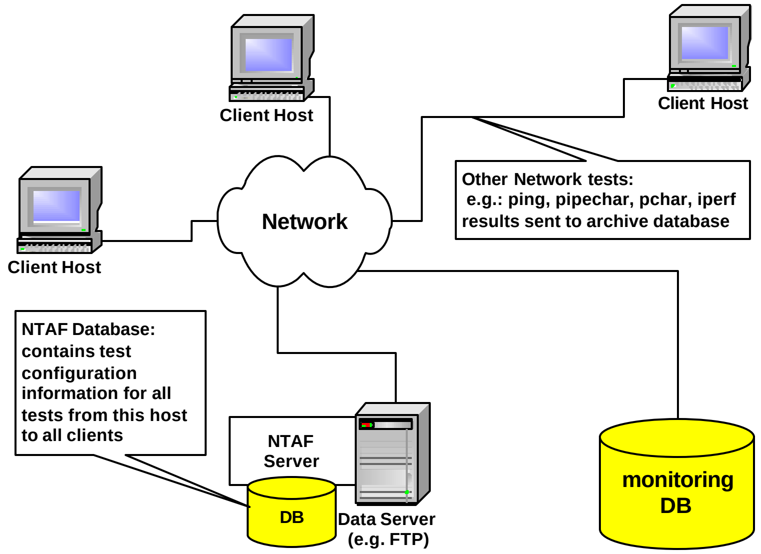
Window (s) 267355.7

Max (s) 1176364

Speed 1.00

Net100 Plans

NTAF Usage



NTAF Use Case

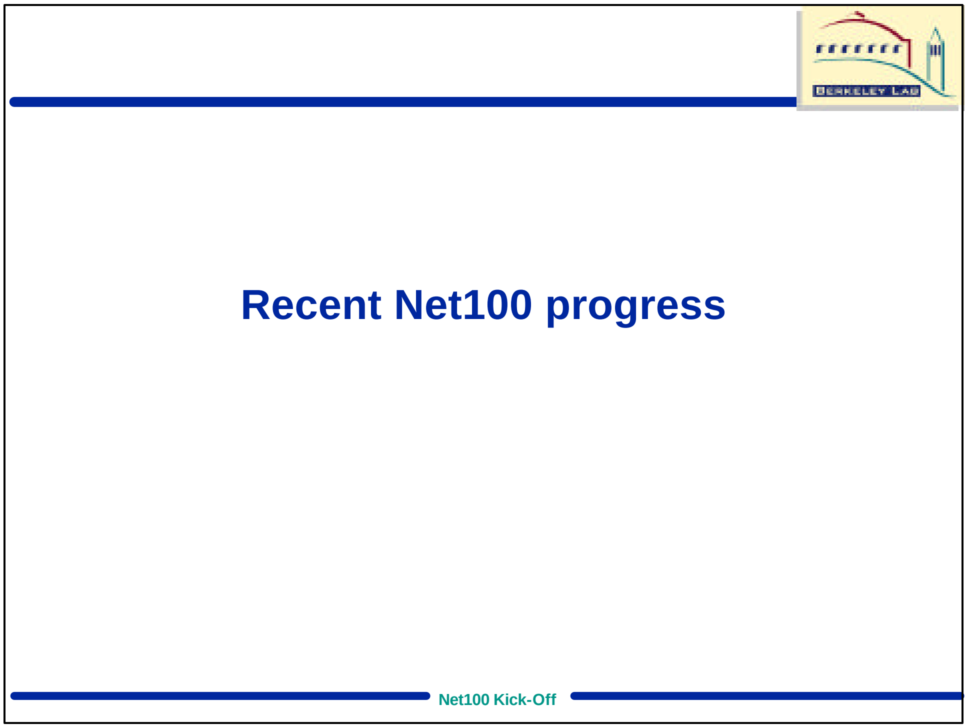
- The NTAF is configured to run the following network tests every few hours over a period of several days:
  - ping -- measure network delay
  - pipechar -- actively measure speed of the bottleneck link
  - iperf -- actively measure TCP throughput. Multiple *iperf* tests could be run with different parameters for the number of parallel streams {e.g.: 1,2,4} and the method of tuning the TCP buffers {Linux 2.4 auto-tuned, hand-tuned}
- All tools will use the Web100 TCP-KIS interface to collect TCP information from the Web100 kernel, and then use NetLogger to format and send this data to the archive.

Use Case (cont.)

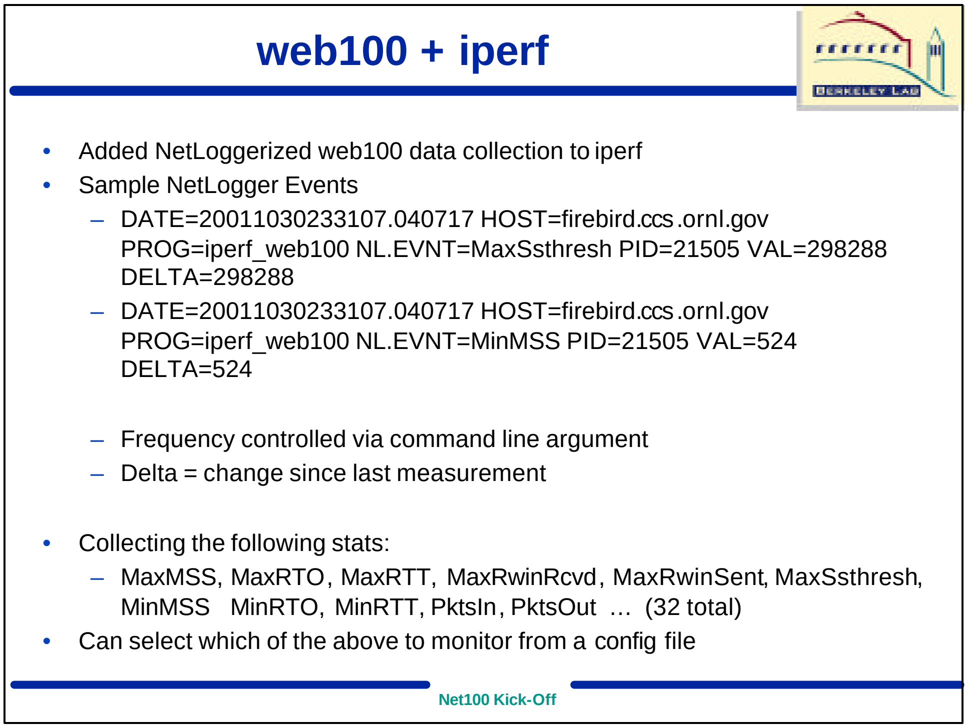
- Analysis based on this test configuration includes the ability to, for ANY path being monitored, do the following:
  - compare Web100 tuned throughput to hand-tuned throughput.
  - compare NWS predicted bandwidth with application and *iperf* bandwidth.
  - determine the advantage, if any, of parallel data streams, using both hand-tuned and autotuned (Linux 2.4-tuned) TCP.
  - see the variability of the results over time.
  - compare pipechar and pathrate to see which is most accurate.
  - measure the impact of tuned TCP streams on non-tuned streams.

Work Plan Year 1

- Begin work on generalizing existing LBNL tools (NetLogger, Enable) to create the core NTAF framework
- instrument iperf, pipechar, and GridFTP using Web100 TCP-KIS and NetLogger
- analyze Linux 2.4 autotuning of GridFTP using NTAF (with ORNL)
- design and implement a simple monitoring archive and interface to the archive



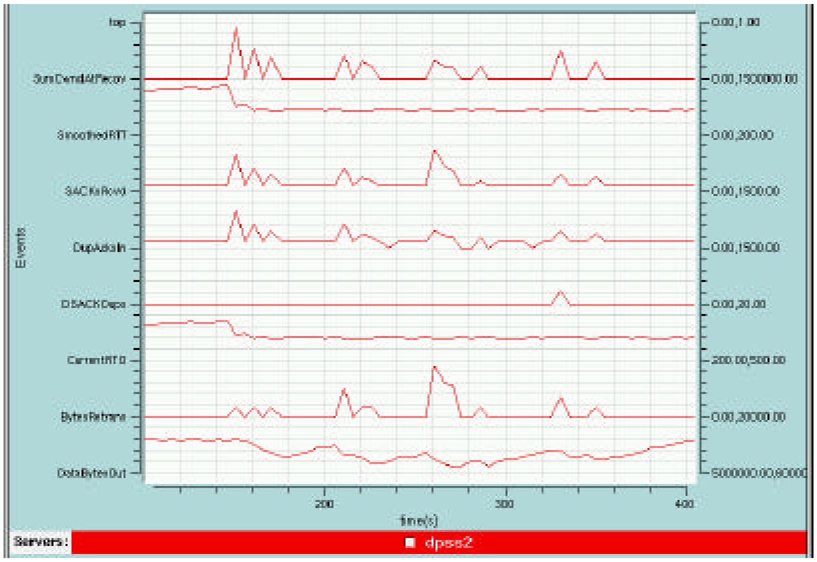
Recent Net100 progress



web100 + iperf

- Added NetLoggerized web100 data collection to iperf
- Sample NetLogger Events
  - DATE=20011030233107.040717 HOST=firebird.ccs.ornl.gov PROG=iperf_web100 NL.EVNT=MaxSsthresh PID=21505 VAL=298288 DELTA=298288
  - DATE=20011030233107.040717 HOST=firebird.ccs.ornl.gov PROG=iperf_web100 NL.EVNT=MinMSS PID=21505 VAL=524 DELTA=524
  - Frequency controlled via command line argument
  - Delta = change since last measurement
- Collecting the following stats:
  - MaxMSS, MaxRTO, MaxRTT, MaxRwinRcvd, MaxRwinSent, MaxSsthresh, MinMSS MinRTO, MinRTT, PktsIn, PktsOut … (32 total)
- Can select which of the above to monitor from a config file

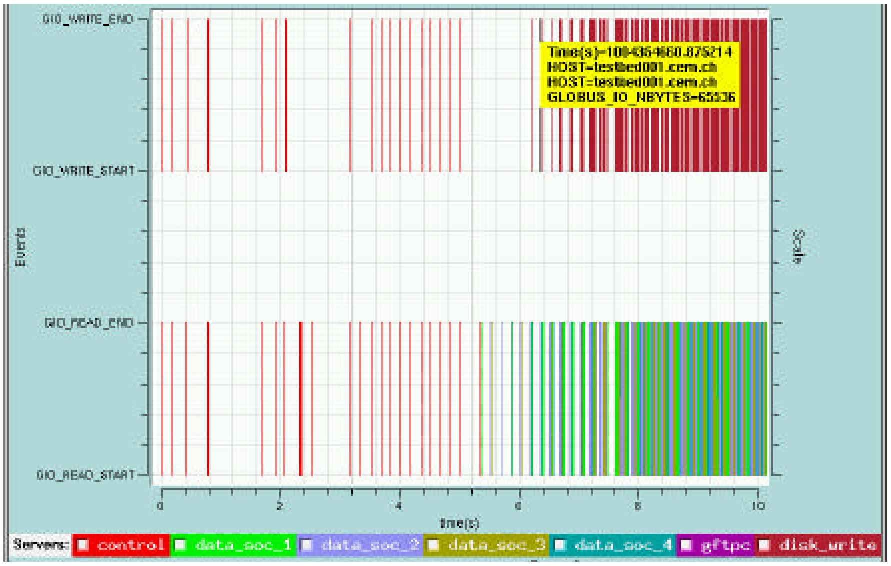
nlv view of iperf web100 data



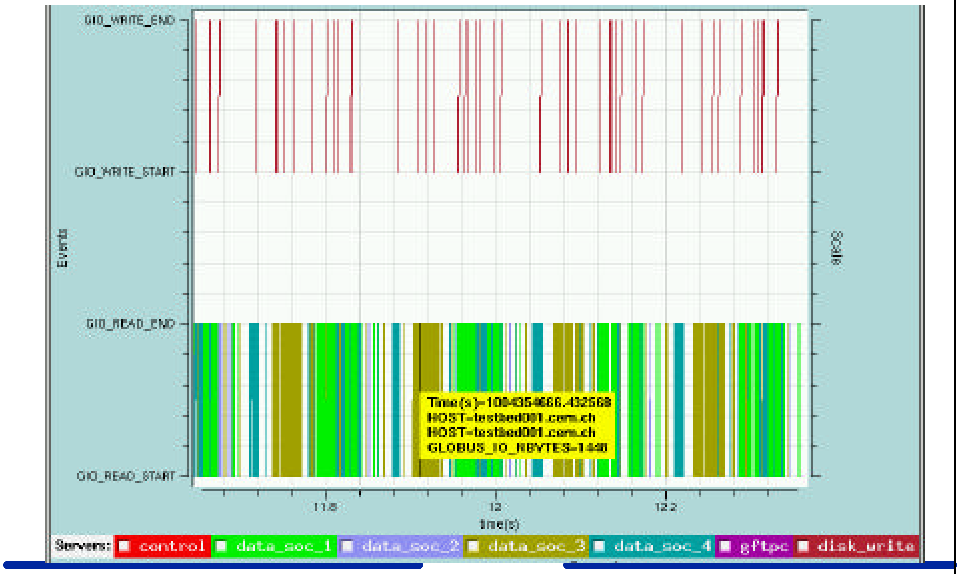
GridFTP + NetLogger

- Worked with Globus developer, John Bresnahan, to add NetLogger to “Globus IO” library
  - Globus GridFTP built on top of Globus IO
  - most Grid projects are using GridFTP
  - just set an environment variable to start logging
- Next step: add web100 KIS support

GridFTP Results: 50MB file

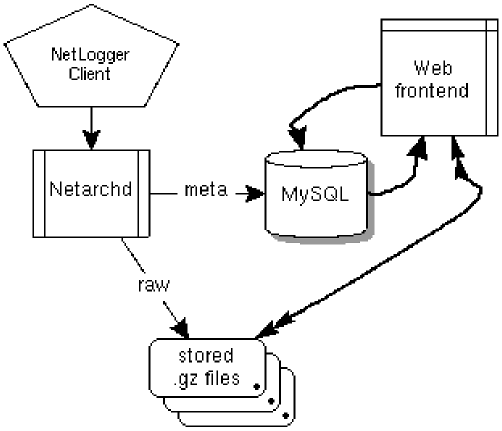


GridFTP results

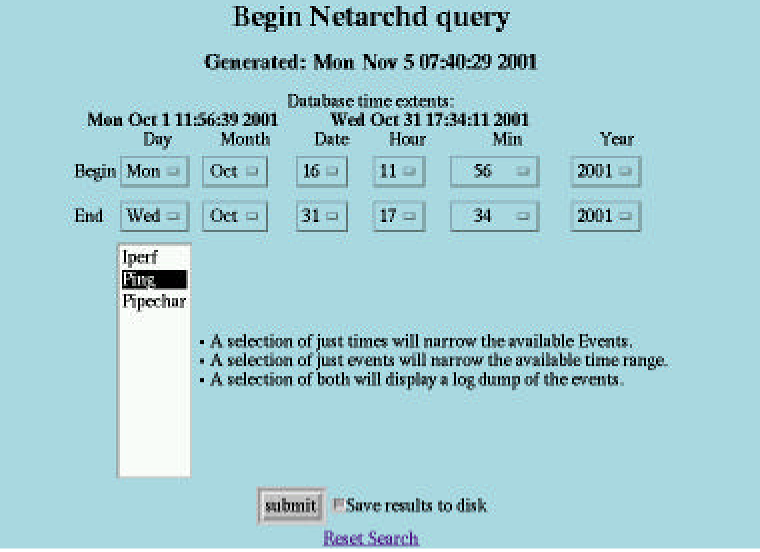


netarchd

- simple mySQL based DB for NetLogger data
- Actual data stored in flat files, DB contains index information only



netarchd Web front-end



Begin Netarchd query

Generated: Mon Nov 5 07:40:29 2001

Database time extents:
Mon Oct 1 11:56:39 2001 **Wed Oct 31 17:34:11 2001**

	Day	Month	Date	Hour	Min	Year
Begin	Mon	Oct	16	11	56	2001
End	Wed	Oct	31	17	34	2001

- A selection of just times will narrow the available Events.
- A selection of just events will narrow the available time range.
- A selection of both will display a log dump of the events.

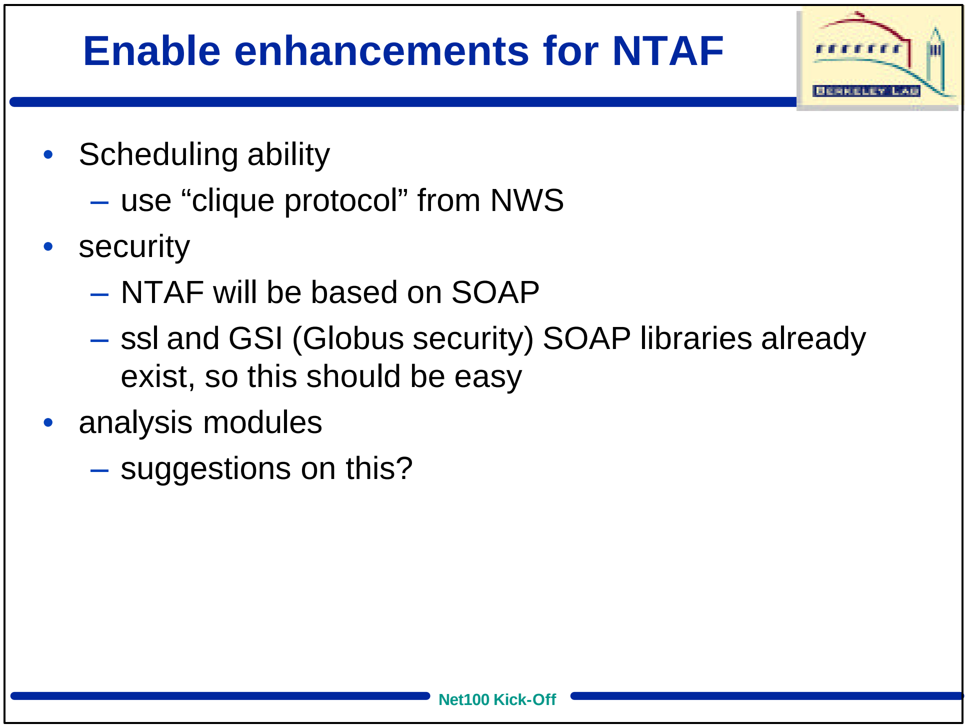
□ Save results to disk

Reset Search

netarchd future work

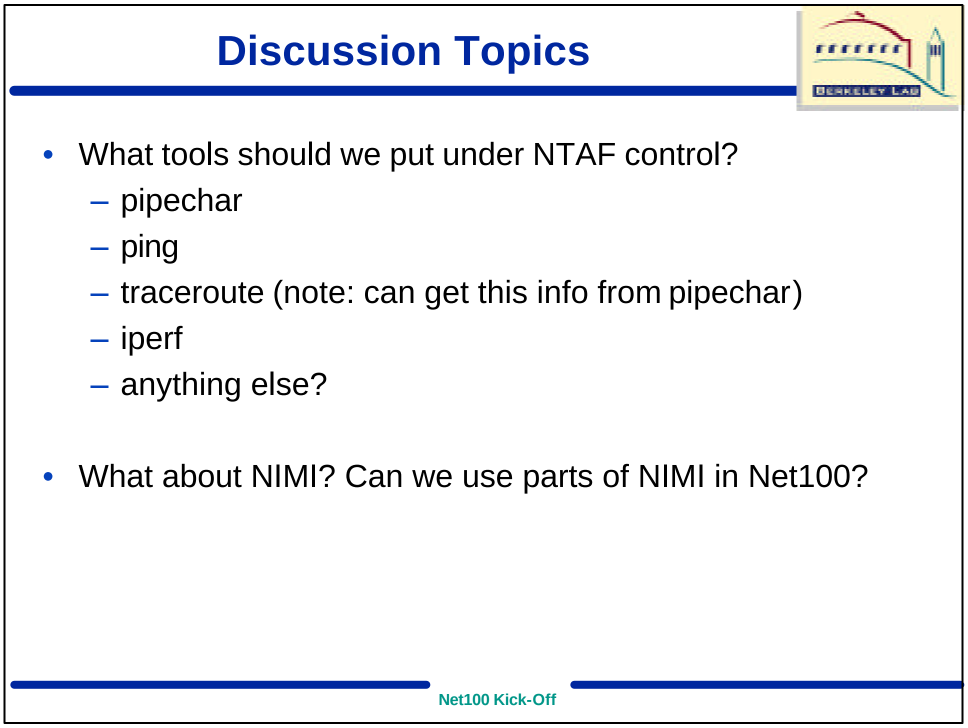
- Security (GSI)
  - currently very little security
  - probably will use GSI-enabled SOAP from LBL
- Allow more complex search queries
- Binary mode for NetLogger clients and netarchd storage.
  - determine scalability limits of netarchd
- output conversion module:
  - binary to ULM or XML or html
  - ULM to XML or html

Discussion Topics



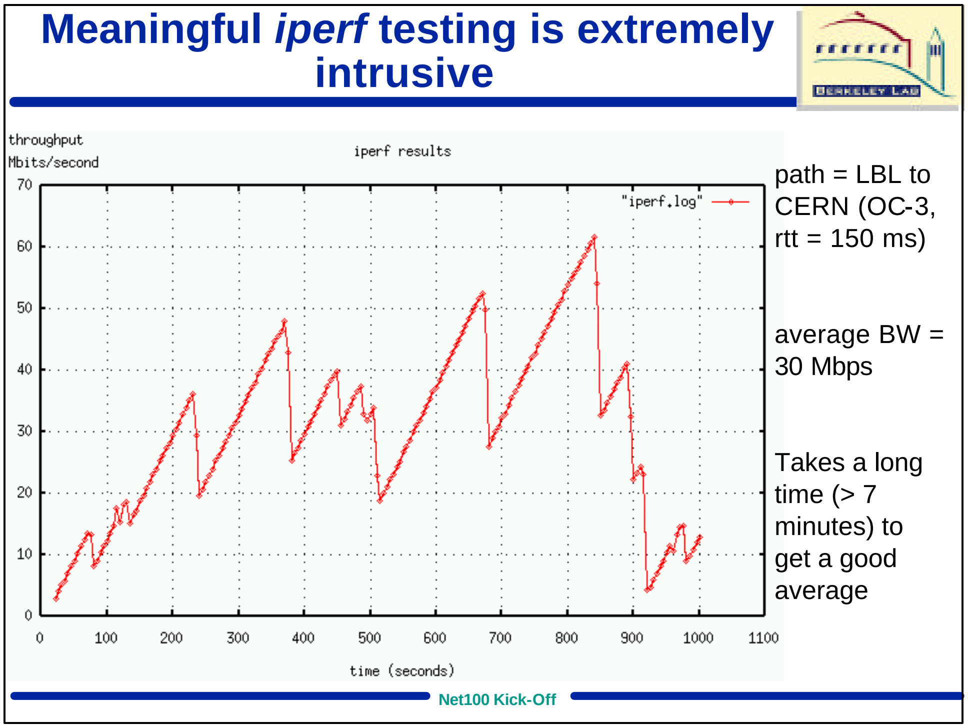
Enable enhancements for NTAF

- Scheduling ability
  - use “clique protocol” from NWS
- security
  - NTAF will be based on SOAP
  - ssl and GSI (Globus security) SOAP libraries already exist, so this should be easy
- analysis modules
  - suggestions on this?



Discussion Topics

- What tools should we put under NTAF control?
  - pipechar
  - ping
  - traceroute (note: can get this info from pipechar)
  - iperf
  - anything else?
- What about NIMI? Can we use parts of NIMI in Net100?



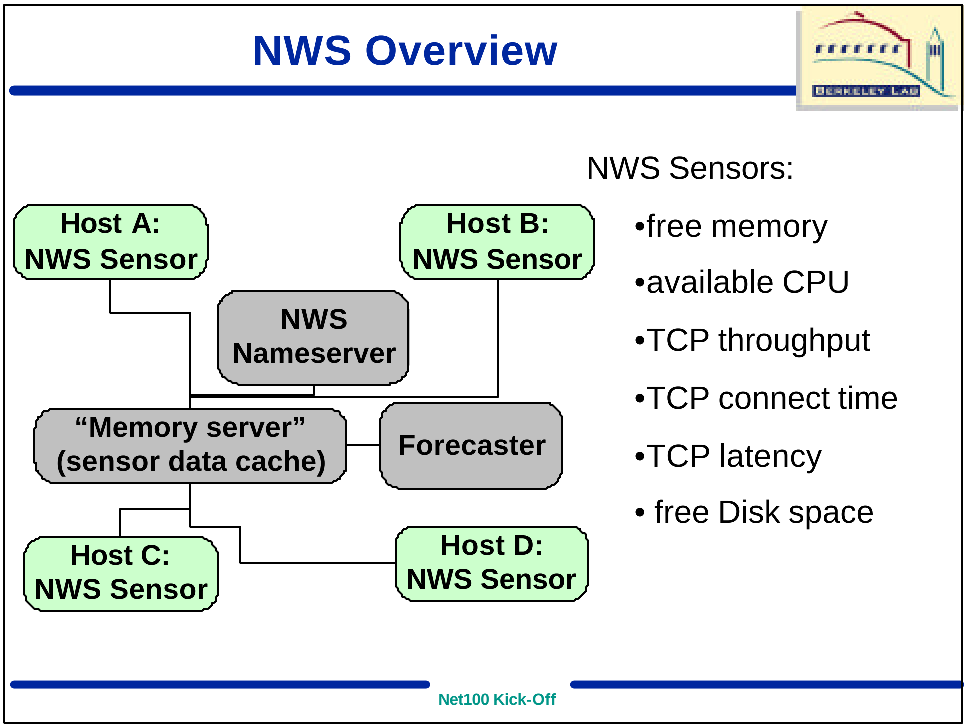
Meaningful *iperf* testing is extremely intrusive

path = LBL to CERN (OC-3, rtt = 150 ms)

average BW = 30 Mbps

Takes a long time (> 7 minutes) to get a good average

Net100 Kick-Off



NWS Overview

Host A: NWS Sensor

Host B: NWS Sensor

NWS Nameserver

"Memory server" (sensor data cache)

Forecaster

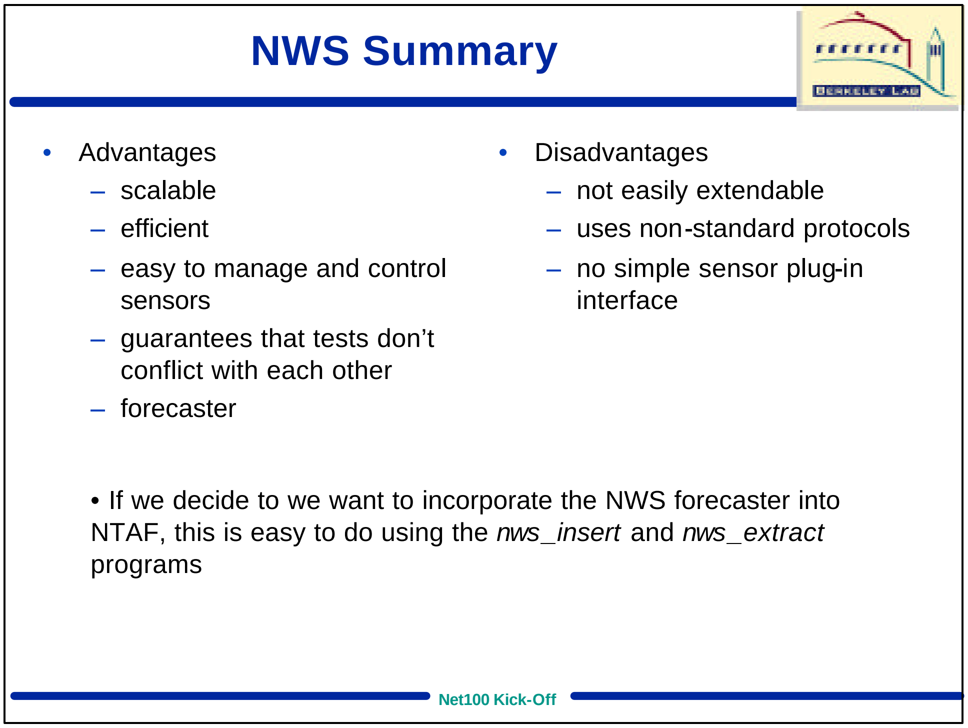
Host C: NWS Sensor

Host D: NWS Sensor

NWS Sensors:

- •free memory
- •available CPU
- •TCP throughput
- •TCP connect time
- •TCP latency
- • free Disk space

Net100 Kick-Off



NWS Summary

- Advantages
  - scalable
  - efficient
  - easy to manage and control sensors
  - guarantees that tests don’t conflict with each other
  - forecaster
- Disadvantages
  - not easily extendable
  - uses non-standard protocols
  - no simple sensor plug-in interface

• If we decide to we want to incorporate the NWS forecaster into NTAF, this is easy to do using the *nws_insert* and *nws_extract* programs